M&M counting software using unsupervised learning algorithms

Camilo Correa Restrepo*

I. Methodology

The aim of this project was, simply put, to construct a simple python script that, when given an image with an arbitrary amount of M&M candies would count how many there were of each color, and would display its results in an easily comprehensible way. With this is mind, the methodology employed will now be explained in detail.

A. Image Retrieval and Preparation

The script, initially, retrieves and loads the image to be processed[1]. Subsequently, the image is converted[2] into the *CIE L*a*b* color space and then split into its constituent channels. Then, the *L* channel is discarded and the *a* and *b* channels are merged. This newly merged matrix, consisting of these channels, is what will be used to construct the feature array. Additionally, a blank image of equivalent size and shape to the original is constructed for later use.

B. Feature Vector Preparation

The matrix that was previously constructed is then iterated upon, pixel by pixel, to create the feature vector, with each *"observation"* consisting of a pixel's *a* and *b* values.

C. K Means Model Preparation and Execution

A *K Means* model was chosen for this implementation, given its ease of use, and relatively low cost in terms of computational power given the simplicity of the feature vector. This model is initialized with a *user-defined*[3] number of clusters, namely, the amount of M&M colors plus the background. This model was then fitted to the previously constructed feature vector. After this, the labels corresponding for each pixel were extracted, as were the *centroids* corresponding to each of the detected colors.

After this computation was complete, the pixel label array was then analyzed and the mode was extracted, in order to identify the color of the background. This operates under the assumption that the most prevalent color that was labeled by the model will be that of the background, as, simply put, it is most likely[4] to be the color that is most present in the original image.

D. M&M Detection

The blank image that was created in the initial stages of processing is then constructed according to the the colors that the model has labeled each pixel with, drawing from the *centroids* that it also calculated. However, all the pixels corresponding to the background are set to black, in order to isolate the collections of pixels that make up the candy.

This image is then converted into *gray-scale*, and then OpenCV's *findContours* function, in conjunction with the found *contour's moments*, are used to determine the position of the candies within the image. The pixel corresponding to the center of the M&M (in the reconstructed image) is then converted into the *CIE L*a*b* color space, and its *a* and *b* values are passed on to the model (using its *predict* function) in order to generate the running count of how many there are within the image of each color. This number is then printed over the corresponding section of the image.

E. Result Presentation

After all this processing is done, a simple image is constructed where all the totals for each color are displayed, in addition to also presenting the reconstructed image showing the count over all the M&Ms. What follows is a presentation of the output that the software generates from one of the test images.

Fig. 1. Original

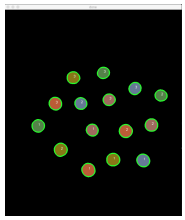

Fig. 2. Processed

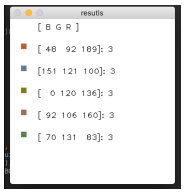

Fig. 3. Results

References

[1] https://docs.opencv.org/
[2] http://scikit-learn.org/stable/documentation.html
[3] https://github.com/mdbloice/Augmentor
[4] http://pandas.pydata.org/pandas-docs/stable/
[5] http://www.numpy.org/

*Estudiante, Ingeniería de Sistemas Universidad EAFIT, código: 201410046010, correo: ccorre20@eafit.edu.co

[1] Unfortunately, it is necessary to manually specify the image path the software, as it isn't equipped or prepared to deal with images in real time.

[2] If the image is of sufficient size, the software's runtime is severely impacted, and, thus, a line of code to resize incoming images has been inserted, though with equivalent ease it is removed. This is not thought to be an issue given the prototypical nature of the software being constructed.

[3] It was infeasible to have the software select the number of clusters in a hands-off manner, as both the *Calinksy-Harabaz Score* and the *Silhouette Score* (which are the only mechanisms that the *sklearn* library present for comparing different number of clusters) were inadequate in dealing with any noise or variation of the background color, and were also deficient even in carefully controlled ideal cases; thus, user intervention was necessary to construct the required functionality.

[4] Obviously, this method would break down if the image were to be sufficiently covered with M&Ms to the point that the background would be mostly obscured, but, for all tested cases this worked as planned.